Multi-objective balancing of assembly lines by population heuristics

Andreas C. Nearchou

Multi-objective balancing of assembly lines by population heuristics

Journal:	*International Journal of Production Research*
Manuscript ID:	TPRS-2006-IJPR-0078.R2
Manuscript Type:	Discussion Note
Date Submitted by the Author:	19-Aug-2006
Complete List of Authors:	Nearchou, Andreas; University of Patras, Department of Business Administration
Keywords:	ASSEMBLY LINE BALANCING, MULTI-CRITERIA DECISION MAKING, HEURISTICS, EVOLUTIONARY ALGORITHMS
Keywords (user):	differential evolution

Multi-objective balancing of assembly lines by population heuristics

Andreas C. Nearchou

Department of Business Administration

University of Patras

26 500 Rio, Patras, Greece

E-mail: nearchou@upatras.gr

Tel.: +30 2610-969980, Fax: +30 2610-996686

Abstract:

This paper is concerned with the solution of the multi-objective single-model deterministic assembly line balancing problem (ALBP). Two bi-criteria objectives are considered: (1) minimizing the cycle time of the assembly line and the balance delay time of the workstations, and (2) minimizing the cycle time and the smoothness index of the workload of the line. A new population heuristic is proposed to solve the problem based on the general differential evolution (DE) method. The main characteristics of the proposed multi-objective DE (MODE) heuristic are: (a) it formulates the cost function of each individual ALB solution as a weighted-sum of multiple objectives functions with self-adapted weights. (b) It maintains a separate population with diverse Pareto-optimal solutions. (c) It injects the actual evolving population with some Pareto-optimal solutions. (d) It uses a new modified scheme for the creation of the mutant vectors.

Moreover, special representation and encoding schemes are developed and discussed which adapt MODE on ALBPs. The efficiency of MODE is measured over known ALB benchmarks taken from the open literature and compared to that of two other previously proposed population heuristics, namely, a weighted-sum Pareto genetic algorithm (GA), and a Pareto-niched GA. The experimental comparisons showed a promising high quality performance for MODE approach.

Key words: assembly line balancing, differential evolution, multi-objective optimization, Pareto optimality, evolutionary algorithms, manufacturing optimization.

1. Introduction

Finding the global optimum to a general multi-objective optimization problem (MOOP) is *NP*-complete (Bäck 1996). Usually, there is no single optimal solution to a MOOP but rather a set of optimal solutions known as Pareto-optimal solutions. These are solutions that are non-dominated by any other solution in the search space when all the objectives are considered, and they do not dominate each other in the set. Due to their intrinsic parallelism, multi-objective evolutionary algorithms (EAs) have recently received a growing research attention for the solution of several real-world MOOPs (Coelo 1999, Van Veldhuizen and Lamont 2000, Jones *et al.* 2002, Tiwari *et al.* 2002).

This paper considers the simple (deterministic single-model) assembly line balancing problem (ALBP) with various objectives. ALBP is a decision problem arising when an assembly line has to be (re)-configured, and consists of determining the optimal partitioning of the assembly work among the workstations in accordance with some objectives (Baybars 1986, Scholl 1999). These objectives usually take one of two forms: either minimizing the number of workstations (m) given the cycle time (c) of the line (SALBP-1), or minimizing c given m (SALBP-2). Any variant of the simple ALBP (SALBP) belongs to the *NP*-hard class of combinatorial optimization problems (Scholl 1999).

Multi-objective (MO) ALB optimization has attracted the research attention in the last decade. In an interesting work, Kim *et al.* (1996), developed a MO genetic algorithm (MOGA) for SALBP with objective to maximize simultaneously the workload smoothness and work relatedness. Ponnambalam *et al.* (2000), applied a MOGA on SALBP with the objective to maximize line efficiency and to minimize workload smoothness index. MOGAs have also developed by (Malakooti and Kumar 1996, Celano *et al.* 1999, Rekiek *et al.* 2001, Chen *et al.* 2002, Mansouri 2005) for solving the general ALBP (GALBP) with various cost and profit oriented objectives. McMullen and Frazier (1998) and Gamberini *et al.* (2006) addressed the stochastic MO GALBP, via simulated annealing, and a special MO heuristic, respectively.

There is a lack in the literature for population heuristics (such as EAs) for solving the multi-criteria SALBP-2. Furthermore, the computational testing of most EAs has been

performed ignoring existing ALB test beds (Scholl and Becker 2006). Inspired in a sense, by the work of Murata *et al.* (1996) who addressed the MO flow-shop scheduling problem via a MOGA; this work presents a new population heuristic based on the differential evolution (DE) method to deal with MO SALBP-2. Two bi-criteria objectives are considered: (1) minimizing the cycle time and the balance delay time of the stations, and (2) minimizing the cycle time and the workload smoothness index of the line. The developed multi-objective DE (MODE) heuristic has the following features:

(a) It formulates the compound objective function of each possible ALB solution as a weighted-sum of the individual objectives functions with self-adapted weights. The proposed self-adaptable scheme for estimating the weights can be very easily adjusted by the decision maker to vary the emphasis on the individual objectives.

(b) It maintains a separate population with diverse Pareto-optimal solutions to the problem. This population is iteratively updated per generation.

(c) It injects the evolving population with elite solutions taken from the Pareto population.

(d) It uses a novel scheme for the creation of the offspring vectors during DE evolution.

MODE's performance is measured over public available benchmarks and compared to that of two known MOGAs proposed by Kim *et al.* (1996) and Murata *et al.* (1996), respectively. The rest of the paper is organized as follows: Section 2 formulates SALBP. Section 3 analyzes the basic DE model for optimization over continuous spaces. Section 4 presents the way DE can be applied on ALBPs, while section 5 introduces MODE for MO SALBP-2. Computational results concerning the performance of the algorithms are provided in section 6, while conclusions and directions for future work are pointed out in section 7.

2. Decision making in ALBPs

2.1. Formulation of SALBP.

SALBP can be stated as follows: m workstations are arranged along an assembly line. Manufacturing a single product on the line requires the partitioning of the total work into a set $V=\{1,\ldots,n\}$ of n elementary operations called tasks. Each task j is performed on exactly one

station and requires a deterministic processing time t_j. Let S_z (z=1,…,m) be the station load of station z (i.e., the set of tasks assigned to z), with a cumulated task time $tS_z = \sum_{j \in S_z} t_j$ (z=1,…,m). The tasks are partially ordered by precedence relations defining a directed acyclic graph (DAG) G=(V,E); with V the set of nodes denoting the tasks in G, and E the set of edges representing the precedence constraints among the tasks. The assembly line is associated with a cycle time c denoting the maximum processing time available for each station. The objective typically takes one of two forms: either minimizing m given c (SALBP-1), or minimizing c given m (SALBP-2). Figure 1 illustrates an example of a precedence graph for an 8-tasks ALBP. The numbers inside the nodes of the graph correspond to the task labels, and those outside the nodes to the processing times.

< *Insert Figure 1 about here* >

In this work, the bi-criteria SALBP-2 is considered with main objective to minimize the cycle time c for a given fixed number of stations m, and secondary objectives to minimize:

(a) The balance delay time (*BD*) of the line (see Eq. (1)). *BD* reflects the unused capacity of the line, i.e., the summation of the idle times of all the stations.

(b) The smoothness index (*SX*) (Eq. (2)) measuring the equality of the distributed work among the stations. The lower the value of *SX* the smoother the line, resulting in reduced in-process inventory. An *SX* equal to zero indicates a perfect balance of the workload among the stations.

$$BD = \sum_{z=1}^{m} \left(c - tS_z\right) \quad \text{`(1)}$$

$$SX = \sqrt{\sum_{z=1}^{m} \left(c - tS_z\right)^2} \quad (2)$$

2.2. The multi-objective SALBP-2

In MO SALBP-2 we ideally seek for a feasible solution that simultaneously optimizes c, as well as, *BD* and *SX*. Since this is almost impossible for any MOOP (Bäck 1996), what we

really attempt to do is to optimize each individual objective to the greatest possible extend. MOOPs considered in this study can be formulated as in the following:

(A) MO SALBP-2 version 1:

Minimize $F_1 = w_1 \cdot c + w_2 \cdot BD$ (3)

subject to:

a partition of the set $V=\{1,\ldots,n\}$ into m disjoint subsets S_z $(z = 1,\ldots,m)$: $\forall$ edge $(i,j)\in E$,

$i, j \in V$ and $j \in FL_i$ the following holds, $i \in S_A$ and $j \in S_B$ with $A \le B$ (3.a)

$tS_z \le t_{sum}$ for all $z = 1,\ldots,m$ (3.b)

where, $t_{sum} = \sum_{j=1}^{n} t_j$ is the sum of all the tasks' processing times and ,

FL_i = the set of immediate successors (followers) of task i.

(B) MO SALBP-2 version 2:

Minimize $F_2 = w_1 \cdot c + w_2 \cdot SX$ (4)

Subject to the constraints (3.a), (3.b)

Constraints (3.a) and (3.b) ensure the feasibility of an ALB solution. In particular, constraint (3.a) guarantees the feasible assignment of the tasks to the m stations. That is, each task is assigned to exactly one station, and the successors of any task i are not assigned to an earlier station than that of i. Note that, (i, j) denotes an edge between i and j, with j being the immediate successor of i. Constraint (3.b) ensure that the station times of all the stations do not exceed the line's total processing time (t_{sum}).

The weights w_1 and w_2 in Eqs.(3),(4), specify the relative importance of the corresponding objectives. The determination of the suitable values for these weights is in general a difficult task and constitutes a critical research question in MOO. This issue will be discussed deeper in section 5 of this study. Moreover, a new methodology will be introduced for a dynamic self-adapted estimation of these weights.

3. Differential evolution (DE)

DE is a population heuristic introduced by Storn and Price (1997) for global optimization over continuous search spaces. DE has been applied with success on many numerical optimization problems outperforming other popular heuristics including GAs (Ali and Törn 2004, Kaelo and Ali 2006). Recently, the application of DE has been extended with success to combinatorial optimization problems with discrete decision variables, such as, the machine layout problem (Nearchou 2006), and machine flow-shop scheduling problems (FSSPs) (Onwubolu and Davendra 2006, Nearchou and Omirou 2006).

DE utilizes Np, D-dimensional parameter vectors $x_{i,k}$, i=1,2,…,Np, as a population to search the feasible region Ω of a given problem. The index k denotes the iteration (or generation) number of the algorithm. The initial population (where, k = 0),

$$\Phi = \{x_{1,0}, x_{2,0}, \ldots, x_{N_p,0}\}, \tag{5}$$

is taken to be uniformly distributed in Ω. At each iteration, all vectors in Φ are targeted for replacement. Therefore, Np competitions are held to determine the members of Φ for the next iterations. This is achieved by using mutation, crossover and acceptance operators. In the mutation phase, for each target vector $x_{i,k}$, i = 1,…,Np, a mutant vector $\hat{x}_{i,k}$ is obtained by

$$\hat{x}_{i,k} = x_{\alpha,k} + F_s \cdot (x_{\beta,k} - x_{\gamma,k}), \tag{6}$$

where $\alpha,\beta,\gamma \in \{1,\ldots,Np\}$ are mutually distinct random indices and are also different from the current target index i. The vector $x_{\alpha,k}$ is known as the base vector and F_s >0 is a scaling parameter. The crossover operator (Eq.(7)) is then applied to obtain the trial vector $y_{i,k}$ from $\hat{x}_{i,k}$ and $x_{i,k}$.

$$y_{i,k}^{L} = \begin{cases} \hat{x}_{i,k}^{L} & \text{if } R^L \le C_R \text{ or } L = I_i \\ x_{i,k}^{L} & \text{if } R^L > C_R \text{ and } L \ne I_i \end{cases}, \tag{7}$$

Where, I_i is a randomly chosen integer in the set I, i.e., $I_i \in I = \{1,2,\ldots,D\}$; the superscript L represents the L-th component of respective vectors; $R^L \in (0,1)$, drawn randomly for each L.

The ultimate aim of the crossover rule is to obtain the trial vector $y_{i,k}$ with components coming from the components of the target vector $x_{i,k}$ and the mutated vector $\hat{x}_{i,k}$. This is, ensured by introducing the crossover rate C_R and the set *I*. Notice that for C_R=1 the trial vector $y_{i,k}$ is the replica of the mutated vector $\hat{x}_{i,k}$. The process (mutation and crossover) continues until all members of Φ are considered. After all *Np* trial vectors $y_{i,k}$ have been generated, acceptance is applied. In the acceptance phase, the cost of the trial vector, $Cost(y_{i,k})$, is compared to $Cost(x_{i,k})$, the value at the target vector and the target vector is updated using

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } Cost(y_{i,k}) < Cost(x_{i,k}) \\ x_{i,k} & \text{otherwise} \end{cases} \quad (8)$$

Mutation, crossover and acceptance phases continue until some stopping conditions are met.

4. Applying DE on ALBPs.

Since DE is quite similar to an EA, without lose the generality, in the following analysis we will use terms borrowed from the field of Evolutionary Computation such as, the *genotype* (is the real-valued vector evolved in DE), the *phenotype* (is the actual ALB solution corresponding to a genotype). Every component of a vector is called a *gene*. Therefore, applying DE on ALBP domain needs the specification of the following five characteristics: (a) a representation mechanism, i.e., a way of encoding the phenotypes to genotypes. (b) An evaluation mechanism, i.e. a way of computing the cost-function for each genotype. (c) A way of initializing a population of genotypes. (d) The application of mutation, crossover and acceptance operators on the population. (e) Values to the parameters: *Np*, C_R and F_s. Characteristics (c)-(e) are in general same as in the standard DE, the differences rely on the way one implement the first two.

4.1. The representation mechanism

The natural coding for ALBPs is strings with integers. Two different schemes for string representation applicable to ALBPs are mainly identified in the literature: station-oriented (*sor*) and task-oriented representation (*tor*) (Scholl and Becker 2006). Both of them assume a string length equal to n (number of the tasks to be processed). Using *sor*, the components of a string are integers in the range [1,m] (m=number of workstations). Thus, if the i-th position of the string has the value z (z =1,…,m), then, task i is assigned to station z. Using *tor*, a specific string is a permutation of the integers 1,2,…,n. Hence, the task j in location i of the string will be assigned to a station before the task in location (i+1) of the string. Note that, a tasks' sequence is legal when it does not break the precedence relations (constraint (3.a)). For example, assume SALBP shown in Fig.1 with m=3, c=28. A possible feasible solution to this problem is to assign tasks {1,2} to station 1, tasks {3,4,6} to station 2, and tasks {5,7,8} to station 3. Using *sor* this solution may be represented by the string <1,1,2,2,3,2,3,3>; while, using *tor* by <1,2,3,4,6,5,7,8>. After experimentation with both schemes over selected test beds (from a set of benchmarks described in section 6), we found *tor* superior, in terms of speed of convergence and quality of solutions, and thus, it was decided to adopt *tor* with DE.

DE works with floating-point vectors and thus an appropriate mapping is needed from the genotypic state-level (the vectors) to the phenotypic level (the actual ALB solutions). To achieve this mapping a simple yet effective topological ordering scheme has been developed based on the relative priorities impose by the components of a genotype. Assuming an n-tasks ALBP with precedence relations given by a DAG $G=(V,E)$, the developed encoding scheme consists of generating a topological sort of G from a specific n-dimensional floating-point vector ψ (genotype). Each vector's component ψ_i (i=1,…,n) represents the relative priority of task i ($i \in V$). The topological sort is therefore a ranking of all the tasks according to their priorities in an appropriate order to meet the precedence constraints. This mechanism is implemented using the following procedure:

Procedure Topological_ordering_encoding
begin
Set $V' = \varnothing$ // with $V' \subseteq V$ //
Repeat
For all $j \in V$ **do**
if j has no predecessors **then** $V' = V' \cup \{j\}$, i.e., insert j into the set V'.
Determine the gene ψ_i of ψ with the maximum value for all $i \in V'$
Insert task i into the next available position in the partial schedule (*PS*).
$V' = V' \setminus \{i\}$, i.e., remove task i from V'.
Until *PS* has been completed
Return *PS*
end

In each step, the tasks with no predecessors are identified and put in set V'. Then, the task in V' having the highest gene's value in ψ is selected, removed from V', and placed in the next available position of *PS*. The process is repeated until the completion of *PS*.

Let us see how this topological ordering works on genotype ψ = (0.32, 0.83, 0.05, 0.24, 0.17, 0.45, 0.09, 0.61) concerning the 8-task ALBP shown in Fig.1. The first position of array *PS* is taken by task 1 (i.e., PS[1]=1) since this is the only task with no predecessors. Task 1 is then cut from DAG and the next task with no predecessors is task 2, thus PS[2]=2. Then, the two tasks 3 and 4 are candidate for the 3rd location of *PS*. The priorities for these tasks are 0.05 and 0.24, respectively, and therefore, *PS*[3]=4 since task 4 has the highest priority, consequently PS[4]=3. Finally, the ALB solution corresponding to ψ will be (1, 2, 4, 3, 6, 5, 7, 8). Fig. 2 displays the detailed step-by-step process for constructing the specific feasible ALB solution. One can see from this figure the partial topological sort, the cut (dark long dashed lines) and the eligible nodes, as well as, the contents of the partial schedule solution *PS*.

< Insert Figure 2 about here >

4.2. Decoding a phenotype into an actual solution for SALBP-2.

Once a specific DEA's genotype is encoded into a feasible ALB solution (a phenotype) then, an appropriate decoding scheme is needed to map this phenotype to an actual solution for SALBP-2. In other words, a method is needed to assign the tasks in the generated task-sequence into the stations. After experimented (over representative instances from the test beds discussed in section 6) with some well known decoding schemes such as the lower and

upper bound search methods (Scholl 1999), we finally decided to adopt a scheme proposed by Kim *et al.* (1996). The use of this scheme (see below) in DE was found to be superior in terms of quality of solutions. The idea is to face SALBP-2 through an iterated procedure that solves the corresponding SALBP-1 with a cycle time value being progressively decreased until reaching a near-optimum value within a specific permitted range.

Procedure decode_SALBP-2
begin
Step 1. Set c initially equal to the theoretical minimum cycle time, i.e., $c = t_{sum}/m$.
Step 2. Assign as many as possible tasks into the first m-1 workstations. Assign all the remaining tasks to the last workstation, m.
Step 3. Calculate the work load W_z for each workstation z (z=1,2,…,m), and the potential workload PW_z (z=1,2,…,m-1) as follows: W_z=the station time tS_z (z=1,2,…,m). $PW_z = tS_z$ + the processing time of the first task assigned to (z+1)st station (z=1,2,…,m-1).
Step 4. Set $c_W = max\ \{ W_1 , W_2 , \ldots, W_m\}$ and $c = min\ \{ PW_1 , PW_2 , \ldots, PW_{m-1} \}$
Step 5. **if** ($c_W > c$) **then goto** step 2 **else** Return c_W
end

4.3. The evaluation mechanism

This mechanism corresponds to the computation of the cost (objective) function for each phenotype solution. As analyzed in sub-section 2.2, the objective is to minimize the functions given by Eqs. (3),(4). Hence, for a phenotype P_i (i=1,…,Np) the cost function is given by,

$$Cost(P_i) = F_j \tag{9}$$

Where, F_j (j=1,2) is the j^{th} composite objective function given by Eqs. (3), (4), respectively.

5. A multi objective DE (MODE) for SALBP-2.

When solving a MOOP often the attempt is to find a Pareto set of optimal solutions. Pareto set contains all those non-dominated solutions to the problem under investigation such that no other solutions are superior to them in respect to all the discrete objectives. To that purpose, MODE tries to find a set of non-dominated ALB solutions rather a single ALB solution. MODE (see Fig. 3 for a schematic overview) has the following two main features:

a) It maintains a separate population of diverse Pareto-optimal ALB solutions iteratively updated generation by generation.

b) It uses an elitist preserving strategy with which a portion of the evolving population is randomly replaced by a number of elite Pareto solutions.

< Insert Figure 3 about here >

In a general MOOP a solution with the best values for each objective can be regarded as an elite solution. Hence, for the bi-criteria ALBPs examined in this work, there are two elite (extreme) solutions in the evolving population each of which optimizes one objective. These solutions are always copied into Pareto population. Pareto set is further completed by additional elite solutions using a mechanism explained below. The Pareto population of the final generation contains the near-optimal solutions to the MOO ALBP. The decision maker can then select that solution accomplishing more her or his preferences.

Moreover, two additional features are included within MODE resulting to modifications to the standard mutation rule given by Eq. (6). These features are as follows:

(i) In each iteration k, mutant vectors $\hat{x}_{i,k}$ (i=1,…,Np) are created using the relation,

$$\hat{x}_{i,k} = r \cdot x_{\alpha,k} + (1-r) \cdot x_{\beta,k} + F_s \cdot (x_{\gamma,k} - x_{\delta,k}) \quad (10)$$

where r is a random number uniformly taken within (0,1) and $\alpha \neq \beta \neq \gamma \neq \delta \neq i \in \{1,\ldots,Np\}$ are distinct random indices. This scheme was found more robust in preliminary experiments than other standard mutant schemes such as the one given by Eq. (6). Particularly, the proposed mutation scheme enhances the ability of the algorithm for converging faster to a near-optimum solution.

(ii) Traditionally, the mutation-factor F_s takes a value within the range (0,2] and this value remains constant through the life cycle of the DE algorithm. In this study, we propose the following dynamic scheme for estimating F_s.

$$if\ Cost_{MIN} \geq 0.95 \times Cost_{AVG}\ \ then\ F_s = F_0\ \ else\ \ F_s = \Theta \times Fs \quad (11)$$

where, $Cost_{MIN}$ and $Cost_{AVG}$ are the population minimum and average costs values, respectively. This scheme gives to F_s a high value at the beginning of the run and decreases this rate slowly by the diversity of the population. F_s is initially defined to be equal to F_0=0.8, and decreased in each new iteration by a factor $\Theta = 0.95$ using the linear relation $F_s = \Theta \times F_s$. If the minimum

population cost becomes almost the same to the average population cost, then a very small diversity is encountered in the population and thus F_s is reset to the initial value F_0. MODE is given below in pseudo-code format:

Algorithm MODE for SALBP-2

Pre-processing step:
Read input data concerning the DAG $G=(V,E)$ of a specific n-tasks ALBP: i.e., the set of tasks V, the set of edges E, the processing times t_j (j=1,…,n), the number of stations m.

Initialization step:
Set values for the control parameters (Np, C_R);
Set mutation scale factor $F_s = F_0$;
Set the size of the Pareto population, Γ_size;
Initialize generation counter $k = 0$;
Generate a population $\Phi = \{x_{1,k}, x_{2,k}, \ldots, x_{Np,k}\}$ of n-dimensional floating-point vectors;
The components of $x_{i,k}$ (i =1,…,Np) are randomly chosen within the range [0,1];
Γ = {∅}; // Create an initial empty Pareto population //

Repeat
for i = 1 **to** Np **do** // create population Φ of the new generation //
Mutation step:
Generate a **mutant** vector $\hat{x}_{i,k}$ using Eq. (10)
Crossover step:
Generate a **trial** vector $y_{i,k}$ by crossing $x_{i,k}$ and $\hat{x}_{i,k}$ using Eq. (7).
Solution Interpretation Step:
// Build the phenotypes corresponding to the genotypes $x_{i,k}$ and $y_{i,k}$ //
$\mathbf{P}^{x_{i,k}}$ = Topological_ordering_encoding ($x_{i,k}$);
$\mathbf{P}^{y_{i,k}}$ = Topological_ordering _encoding ($y_{i,k}$);
// Decode phenotypes to actual solutions for SALBP-2 using the scheme presented in sub-section 4.2. Then evaluate the cost of each solution using Eq.(9) //
if MO SALBP-2 version 1 **then**
$Cost(\mathbf{P}^{x_{i,k}}) = F_1$ (decode_SALBP-2($\mathbf{P}^{x_{i,k}}$))
$Cost(\mathbf{P}^{y_{i,k}}) = F_1$ (decode_SALBP-2($\mathbf{P}^{y_{i,k}}$))
else if MO SALBP-2 version 2 **then**
$Cost(\mathbf{P}^{x_{i,k}}) = F_2$ (decode_SALBP-2($\mathbf{P}^{x_{i,k}}$))
$Cost(\mathbf{P}^{y_{i,k}}) = F_2$ (decode_SALBP-2($\mathbf{P}^{y_{i,k}}$))
endif
Acceptance step:
if $Cost(\mathbf{P}^{y_{i,k}}) < Cost(\mathbf{P}^{x_{i,k}})$ **then** $x_{i,k+1} = y_{i,k}$ **else** $x_{i,k+1} = x_{i,k}$
endfor
Update Pareto Population Step:
// Check each one of the individual solutions in Φ whether constitutes a Pareto solution //
cΦ = cΓ = 0 ; // initialize counters for the members in Φ and Γ, respectively //
While (cΓ ≤ Γ_size) and (cΦ ≤ Np) **do**
cΦ = cΦ + 1;
Compare Φ(cΦ) (i.e., the cΦ member of Φ) with all Pareto solutions in Γ;

If Φ(cΦ) is not contained in Γ **then**
If it dominates some Pareto solutions **then**
Add Φ(cΦ) into Γ and delete the solutions dominated by it;
Increment accordingly counter $c\Gamma$;
else if there is empty space in Γ **then**
Add Φ(cΦ) into Γ.
$c\Gamma = c\Gamma + 1$;
endif
endif
endwhile
Elitist Preserving Strategy Step:
Determine the two elite Pareto solutions in Γ;
Randomly select two members in Φ and replace them with the two elite Pareto solutions from Γ;
Adaptation of parameter F_s **Step:**
Determine worst and average cost functions in Φ; then, adapt F_s using Eq. (11);
$k = k + 1$ // increment iteration counter //
Until $k >$ MAXI; // MAXI stands for Maximum Iterations //
Return Γ ;

As it was referred in sub-section 2.2, weighted-sum method is used to construct the composite objective functions F_1 and F_2 given by Eqs.(3) and (4), respectively. In the literature, there are two general methods to compute the weights w_i (i=1,…,Q) for a weighted-sum objective function with Q objectives: the fixed-weight method and the random-weight method. The former uses constant weights satisfying the relation,

$$\sum_{i=1}^{Q} w_i = 1 \qquad w_i > 0 \ \text{for all} \ i = 1,\ldots,Q \tag{12}$$

However, as Murata *et al.* (1996) shown, using constant weights within an EA the search direction is fixed, and for this reason it is difficult for the search process to obtain a variety of non-dominated solutions. To overcome this drawback, Murata *et al.* (1996), proposed the use of random weights according to the following formula,

$$w_i = \frac{random_i}{\text{random}_1 + random_2 + \cdots + random_Q} \qquad i = 1,2,\ldots,Q \tag{13}$$

where $random_i$ (i=1,…,Q) are non-negative random numbers.

Furthermore, Gen and Cheng (2000) proposed an adaptive weight approach within a MOGA, which readjusts the weights by utilizing some useful information from the current population. This method computes the weights by,

$$w_i = \frac{1}{z_i^{\max} - z_i^{\min}}, \quad \text{for all } i = 1,\ldots,Q \tag{14}$$

where $z_i^{\max}$ and $z_i^{\min}$ are the maximal and minimal values of the i^{th} objective in the population.

In this work, a new, self-adapted method for the estimation of the weights w_1 and w_2 (used in Eqs.(3),(4)) has been developed. The proposed method is given by the following relation:

$$\begin{aligned} w_1 &= \lambda \cdot e^{-|d|} + \mu \\ w_2 &= 1 - w_1 \end{aligned} \tag{15}$$

where, λ and μ are user-defined coefficients used to normalize the upper and lower bounds of w_1, respectively. In this study, we set $\lambda = \mu = 0.5$. The exponent d, is determined by,

$$d = c^* - c \tag{16}$$

c^* is the theoretical minimal cycle time of the assembly line. Note that the proposed scheme gives to the first objective higher priority assuming that this is the most significant criterion in the composite objective function. In SALBP-2 the main criterion is to minimize c, hence, w_1 is increased while d approaches zero, i.e., when a generated ALB solution approaches the optimal solution regarding the cycle time criterion. Figure 4 shows the rate of change of the w_1 in respect to the changes of parameter d. It is worth pointing that, the application of the proposed scheme on any bi-criteria optimization problem is straightforward provided by the decision maker the main and the secondary optimization criteria.

< Insert Figure 4 about here >

6. Numerical Results and discussion

6.1 Experimental Setup

MODE was compared against two representative MOGAs identified in the literature: the first is a Pareto-niched GA proposed by Kim *et al.* (1996) for solving MO SALBP, while the second is a Pareto weight-sum GA developed by Murata *et al.* (1996) to address MO FSSP. We will refer to these algorithms with the abbreviations MOGA1 and MOGA2, respectively. Although MOGA2 was introduced for FSSP, as will be explained below, very easily it can be extended and applied on SALBP, as well. Three versions of MODE were implemented each one differ on the way the weights in the objective function are estimated. The first version uses fixed and equal value weights ($w_1 = w_2 = 0.5$), the second version uses random weights estimated by Eq. (13), and the third version uses the proposed adaptive scheme given by Eq. (15). In the following analysis we will refer to the three MODEs as, MODE1 (with fixed weights), MODE2 (random weights), and MODE3 (adaptive weights), respectively.

All the heuristics were implemented in Delphi Pascal and run on a Pentium 4 (1.7 GHz) PC. The experiments were carried out on known ALBP benchmarks taken from the open literature (Scholl 1999). Note that the upper bounds on the optimal objective function values for these benchmarks concern the minimal cycle time. In the experiments we include the available test instances concerning the following five ALBPs: Buxey (n=29, in#=8), Sawyer (n=30, in#=8), Gunther (n=35, in#=10), Kilbridge (n=45, in#=9), and Tonge (70, in#=23). n denotes the number of the tasks in the corresponding precedence graphs and in# the number of the test instances included in the specific ALBP. To be fair with the stochastic behavior of the five heuristics, we run each one of them ten times over every test instance and the solutions quality were averaged. This means that, each heuristic was run over (8+8+10+9+23)×10=580 test experiments in total. All the examined heuristics were defined to evolve a fixed size population of 2n individual solutions, and run for a maximum of 100n iterations.

Remark 1: MOGA1 combines a Pareto GA and a niched GA. Ranking is performed using the Goldberg's ranking method (Goldberg 1989). This method assigns to Pareto solutions the same rank and to all the others solutions a some less desirable rank. The niche radius in the shared fitness function was defined to be equal to $1/\sqrt{popsize}$, with *popsize* denoting the population size. Chromosomes are integer strings and are mapped into ALB solutions through sequence-oriented representation. If a task sequence breaks the precedence constraints then a suitable repairing method is applied to correct it. Crossover and mutation are performed through the partially mapped (PMX) crossover and reciprocal exchange, respectively. The rates of crossover and mutation operators were set (same as in Kim *et al.* (1996)) equal to 0.3 and 0.5, respectively. The repairing procedure is also applied after the application of these operators on the chromosomes. Selection is done by a tournament strategy (Goldberg 1989).

Remark 2: In MOGA2, a chromosome is a permutation of the integer numbers 1,2,…,*n*, with each gene in the chromosome denoting a different task label. A scalar fitness function is used formulated as a weighted sum function with the weights estimated using Eq. (13). For each chromosome x_i, i=1,2,…,*popsize* the probability of being selected for reproduction is given by the ratio $\left(f(x_i)-f_{\min}\right)\Big/\left(\sum_{i=1}^{popsize}(f(x_i)-f_{\min})\right)$. Where, $f(x_i)$ is the fitness function of x_i and $f_{\min}$ the minimum population fitness function. Once these selection probabilities are estimated for all the vectors in the entire population, selection is performed using the roulette wheel strategy (Goldberg 1989). Crossover is performed by a two-point crossover procedure with a rate equal to 1.0, while mutation is performed via the shift mutation operator with a rate 1/*n*. To map a chromosome to a feasible ALB solution we used the same representation mechanism, as well as, the same repairing method as in MOGA1. The tasks in an ALB solution are assigned to the stations according to the scheme described in sub-section 4.2. MOGA2 maintains a separate set with Pareto solutions, and applies an elitist strategy with these solutions on the entire population.

6.2 Choice of the control parameters' settings for MODE

Much investigation on the selection of the appropriate settings of the control parameters (population size $Np \geq 4$, crossover rate $C_R \in [0,1]$, and mutation-scale factor $F_s \in (0,2]$) was undertaken in preliminary tests. Here, we describe the experimental design methodology used to determine these settings.

In particular, two levels of $Np \in (n, 2n)$ were examined. C_R was defined to take values within the discrete range {0.1, 0.3, 0.5, 0.7, 0.9}, while for F_s two different control schemes were used: (a) a static scheme based on which F_s takes values in the range [0.5, 0.7, 0.9, 1.2], and (b) the proposed dynamic scheme given by Eq. (11).

< *Insert Table 1 about here* >

< *Insert Table 2 about here* >

After 50 runs of MODE algorithm over representative ALBP instances with the above control schemes we determined that the best results obtained were due to the combination $Np=2n$, $C_R=0.7$ and F_s being estimated by Eq.(11). Table 1 presents the effect of C_R, and F_s (when $Np=2n$) on MODE's performance, over Buxey's instances. For each different pair (F_s ,C_R) two numbers are reported in this table. The first number corresponds to the mean percentage deviation of the generated cycle time from the existing optimum solution, and the second number enclosed in brackets corresponds to the mean % effort (see Eq (17)) spent by the algorithm until attained this solution.

$$\%\ effort = \frac{K_{opt}}{MaxK} \times 100 \qquad (17)$$

where K_{opt} is the number of iterations attained the best solution and $MaxK=100n$ is the maximum permitted number of iterations. As one can see from Table 1, the best results are due to (F_s ,C_R)=(adaptive**,** 0.7). With these settings MODE achieved ALB solutions with a mean deviation from the global optimum equal to 0.27% after spending approx. the 34% of the total permitted iterations. Hence, in the following discussion, all the experimental results obtained by MODE heuristics are due to the above 'optimal' combination of settings.

6.3 Comparison of MODE and MOGA heuristics

Tables 2 and 3 display the comparative results obtained by the five heuristics over the benchmarks instances described in sub-section 6.1. The results in Table 2 concern the experiments with the objective given by Eq.(3), while the results presented in Table 3 concern the experiments with the objective given by Eq.(4). The first and second columns of Tables 2 and 3 indicate the problems tested and their size, respectively, while the third column of the tables indicates the method used. The rest columns of the tables provide the following information:

- **c%dev** = the average relative deviation from optimum in percentage; estimated by $((c-c^*)/c^*)\times 100$, with c^* the existing optimal cycle time, and c the cycle time of the best solution generated by a specific heuristic.
- **BD** = the average balance delay time corresponding to the optimal solution attained by a heuristic (this information is included in Table 2).
- **SX** = the average smoothness index of the optimal solution attained by a heuristic (included in Table 3).
- **Compound cost** = includes the best, worst, and mean population costs (Eq. (9)) generated by each heuristic.
- **%effort** = denotes the convergence ratio of a heuristic in % given by Eq. (17).
- **cpu-time** = the average actual processing time in seconds spent by each heuristic until the convergence to the best possible solution.

< Insert Table 3 about here >

As one can observe from Table 2 the best results concerning c%dev have been obtained by MODE3 with a mean offset from optimum approx. equal to 0.27% (BD≈15) for Buxey's problems, 1.56% (BD≈18) for Sawyer's problems, 0.25% (BD≈44) for Gunther's problems, etc. The second best performance is reported by MODE1 with (c%dev, BD) approx. equal to (0.71%, 18) for Buxey's problems, (1.92%, 16) for Sawyer's problems, and so on. Similar high performance for MODE1 and MODE3 heuristics is reported in Table 3. Again, the

proposed MODE3 outperformed all the other heuristics generating solutions of higher quality in respect to both objectives, either c%dev, or SX. Near to this performance stays that attained by MODE1. Furthermore, a significant observation from these tables is that the use of SX as the second optimization criterion in the objective function results in much higher quality solutions (independently of the heuristic used) than those obtained when using BD as second optimization criterion. Another observation from Tables 2 and 3 is that using random weights in the compound objective function of the MODE heuristic (case of MODE2) results to a lower quality performance in comparison to the other heuristics. In regard to the cpu-time spent by the heuristics, the two MOGAs are in general faster than MODEs heuristics (see last column in Tables 2 and 3, respectively) with MOGA2 being the fastest.

< Insert Figure 5 about here >

For better illustration of the generated results we built Figures 5 and 6. In particular, Fig. 5(a) shows graphically the fluctuation of c%dev over the five ALBPs when BD is used as second optimization criterion. The associated fluctuation of the average BD values is shown in Fig. 5(b). It is clear from Fig. 5 that MODE3 is superior from the other heuristics. Fig. 6 shows the fluctuation of the mean c%dev (Fig. 6(a)) and the associated average SX values (Fig. 6(b)) in regard to the compound objective given by Eq. (4).

< Insert Figure 6 about here >

Finally, Fig. 7 shows the %effort spent by the five heuristics over the various benchmarks in regard to the objectives of Eq. (3) (Fig. 7(a)) and Eq. (4) (Fig. 7(b)). As one can see from these figures, MODE3 (the lowest curve) spends in average less iterations until the convergence to the near-optimum solution than the other heuristics.

< Insert Figure 7 about here >

7. Conclusions

Any variant of the simple assembly line balancing problem (SALBP) is NP-hard and thus, it is justified to address large-size instances of the problem through the use of heuristics. This paper introduced a multi-objective (MO) differential evolution (DE) based approach for

solving the bi-criteria SALBP. The main objective was to minimize the cycle time of the line and secondary objectives to minimize balance delay time and workload smoothness index. MODE differs from existing MO population heuristics in at least four features: the encoding scheme used to represent the feasible ALB solutions, the evaluation mechanism to compute the multiple objectives, the procedure for generating the new individual solutions, and in the way it seeks and maintains the set of the non-dominated solutions. Particularly, MODE has the following main characteristic: (a) utilizing a robust encoding scheme maps real-valued vectors (genotypes) to integer strings corresponding to feasible ALB solutions (phenotypes). (b) Every objective function assigning a cost value to a genotype is formulated as a weighted-sum of the individual objectives with the weight coefficients being dynamically adjusted by a new efficient method. The application of this method on any bi-criteria optimization problem is straightforward provided by the decision maker (DM) the main and the secondary optimization criteria. (c) Mutant vectors are generated using a modified efficient strategy. (d) It maintains and updates iteratively a set of non-dominated solutions separately of the actual evolving population, as an attempt to obtain quality and diverse Pareto-optimal solutions. (e) It uses an elitist strategy to preserve non-dominated solutions found over generations from getting lost.

MODE is simple, and very easily implemented. Extensive experimental comparisons over public available ALB benchmarks between MODE and two existing MO evolutionary algorithms showed a superior performance for the former in terms of quality of solutions.

In practice, many MO optimization problems (MOOPs) have multiple conflicting objective functions expressed in differing units, and with an inverse, nonlinear relationship among themselves. These objectives may be even imprecise (or fuzzy) in nature to be defined. In its present form MODE cannot address such problems. Hybridizing MODE with mechanisms borrowed from the field of nonlinear goal programming (GP) (Lee 1972) may result to a promising optimization tool for these problems. GP needs DM to provide a numeric goal (together with a priority level) for each objective, and then seeks for a solution that minimizes the weighted-sum of the deviations of the objective functions from their respective

goals. This idea will constitute a central research direction in the near future. Moreover, there are MOOPs with a huge set of Pareto-optimal solutions for which evaluating this set to select the best one becomes unpractical for DM. Perhaps, a solution for these problems can be obtained by trying to get compromises, based on the DM's information. Compromise solution-based fitness assignments (Gen and Chen 2000), is an interesting approach to be investigated within MODE.

Moreover, this work is limited in the deterministic single-model ALBP, however, it represents a good start point for further studies focused on more difficult ALBPs such as the stochastic or dynamic ALBP. In reality, tasks' processing times are rarely deterministic and may vary more or less. When these variations are considerable then we have the stochastic ALBP. Dynamic ALBP considers operation times being varying over time, e.g. due to learning effects, or successive improvements of the production process (Scholl 1999). Our intuition is that, MODE can be rather easily extended to address the stochastic SALBP provided that a suitable statistical model will be developed that transforms the stochastic task times to deterministic ones, and realized different cycle times so that to avoid blocking and starving of the workstations.

Another avenue for further research is to consider the mixed-model ALBP (MALBP). This problem is much more complex than SALBP since, the attempt is to manufacture different versions (models) of the same basic product in the same line (e.g., PCs with or without DVD drive, with various CPU types, etc.) in arbitrarily intermixed sequence. A first idea is to address the feasibility MALBP; i.e., given the cycle time c and the number m of the stations determine whether or not, a feasible mixed-model assignment with m stations exist.

Acknowledgments

The author thanks the anonymous referees for their valuable comments and suggestions on this paper. This work is integrated in the Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

References

Ali M.M. and Törn A. 2004, Population set-based global optimization algorithms: some modifications and numerical studies, Computers & Operations Research, **31**, 1703-1725.

Bäck T. (1996), Evolutionary algorithms in theory and practice. Oxford University Press, New York.

Baybars I., 1986, A survey of exact algorithms for the simple assembly line balancing problem, Management Science, 32, 909-932, 1986.

Celano G., Fichera S., Grasso V., Commare U. Perrone G. 1999, An evolutionary approach to multi-objective scheduling of mixed model assembly lines, Computers & Industrial Engineering, 37, 69-73.

Chen R.-S., Lu K.-Y., and Yu S.-C., 2002, A hybrid genetic algorithm on multi-objective assembly planning problem, Engineering Applications of Artificial Intelligence, 15, 447-457.

Coelo C.A.,1999, A comprehensive survey of evolutionary-based multiobjective optimization, Knowledge and Information Systems, 1/3, 129-156.

Gamberini R., Grassi A., and Rimini B., 2006, A new multi-objective heuristic algorithm for solving the stochastic assembly line re-balancing problem, Int. Journal of Production Economics, 102/2, 226-243.

Gen M. and Cheng R. 2000, Genetic algorithms and engineering optimization, A Wiley-Interscience publication, New York.

Goldberg D.E. 1989, Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA.

Jones D.F., Mirrazavi S.K, and Tamiz M. 2002, Multi-objective meta-heuristics: An overview of the current-state-of-the-art, European Journal of Operational Research, 137, 1-9.

Kaelo P. and Ali M.M., 2006, A numerical study of some modified differential evolution algorithms, European Journal of Operational Research, 169(3), 1176-1184.

Kim Y.K., Kim Y.-J., and Kim Y., 1996, Genetic algorithms for assembly line balancing with various objectives, Computers Industrial Engineering, 30/3, 397-409.

Lee S., 1972, Goal programming for decision analysis, Auerbach Publishers, Philadelphia.

Malakooti B. and Kumar A., 1996, A knowledge-based system for solving multi-objective assembly line balancing problems, Int. Journal of Production Research, 34(9), 2533-2552.

Mansouri S.A., 2005, A multi-objective genetic algorithm for mixed-model sequencing on JIT assembly lines, European Journal of Operational Research, 167, 696-716.

McMullen P.R. and Frazier G.V., 1998, Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations, Int. Journal of Production Research, 36/10, 2717-2741.

Murata T., Ishibuchi H., and Tanaka H., 1996, Multi-objective genetic algorithms and its application to flowshop scheduling, Computers and Industrial Engineering, 30/4, 957-968.

Nearchou A.C., 2006, Meta-heuristics from nature for the loop layout design problem, Int. Journal of Production Economics, 101/2, 312-328.

Nearchou A.C and Omirou S.L., 2006, Differential evolution for sequencing and scheduling optimization, Journal of Heuristics, (to appear).

Onwubolu G.O. and Davendra D., 2006, Scheduling flow shops using differential evolution, European Journal of Operational Research, 169, 1176-1184.

Ponnambalam S. G., Aravindan P, and Naidu M.G., 2000, A multi-objective genetic algorithm for solving assembly line balancing problem, Int. Journal of Advanced Manufacturing Technology, 16, 341-352.

Rekiek B., De Lit P., Pellichero F., L'Englise T., Fouda P., Falkenauer E., and Delchambre A., 2001, A multiple objective grouping genetic algorithm for assembly line design, Journal of Intelligent Manufacturing, 12, 467-485.

Scholl A.,1999, Balancing and sequencing of assembly lines, Physica-Verlag publ., Heidelberg, Germany.

Scholl A. and Becker C., 2006, State of the art exact and heuristic solution procedures for simple assembly line balancing, European Journal of Operational Research, 168(3), 666-693.

Storn R. and Price K., 1997, Differential Evolution–A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, 11(4), 341-354.

Tiwari A., Roy R., Jared G., and Minaux O., 2002, Evolutionary-based techniques for real-life optimization: development and testing, Applied Soft Computing, 1, 301-329.

Van Veldhuizen D. A. and Lamont G.B., 2000, Multiobjective evolutionary algorithms: Analyzing the state-of-the-art, Evolutionary Computation, 8(2), 125-147.

List of Figures

List of Tables

Figure 1

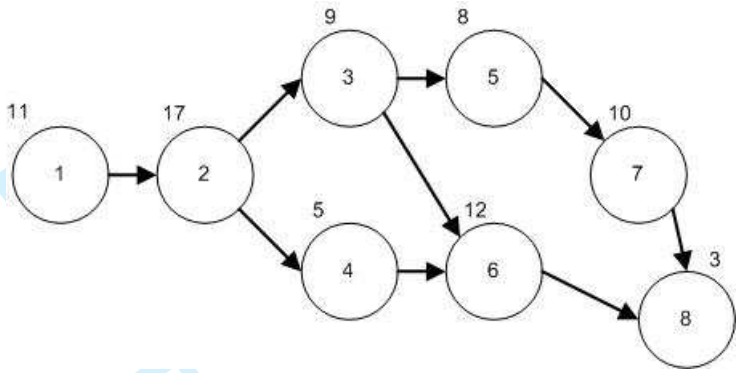


Figure 2

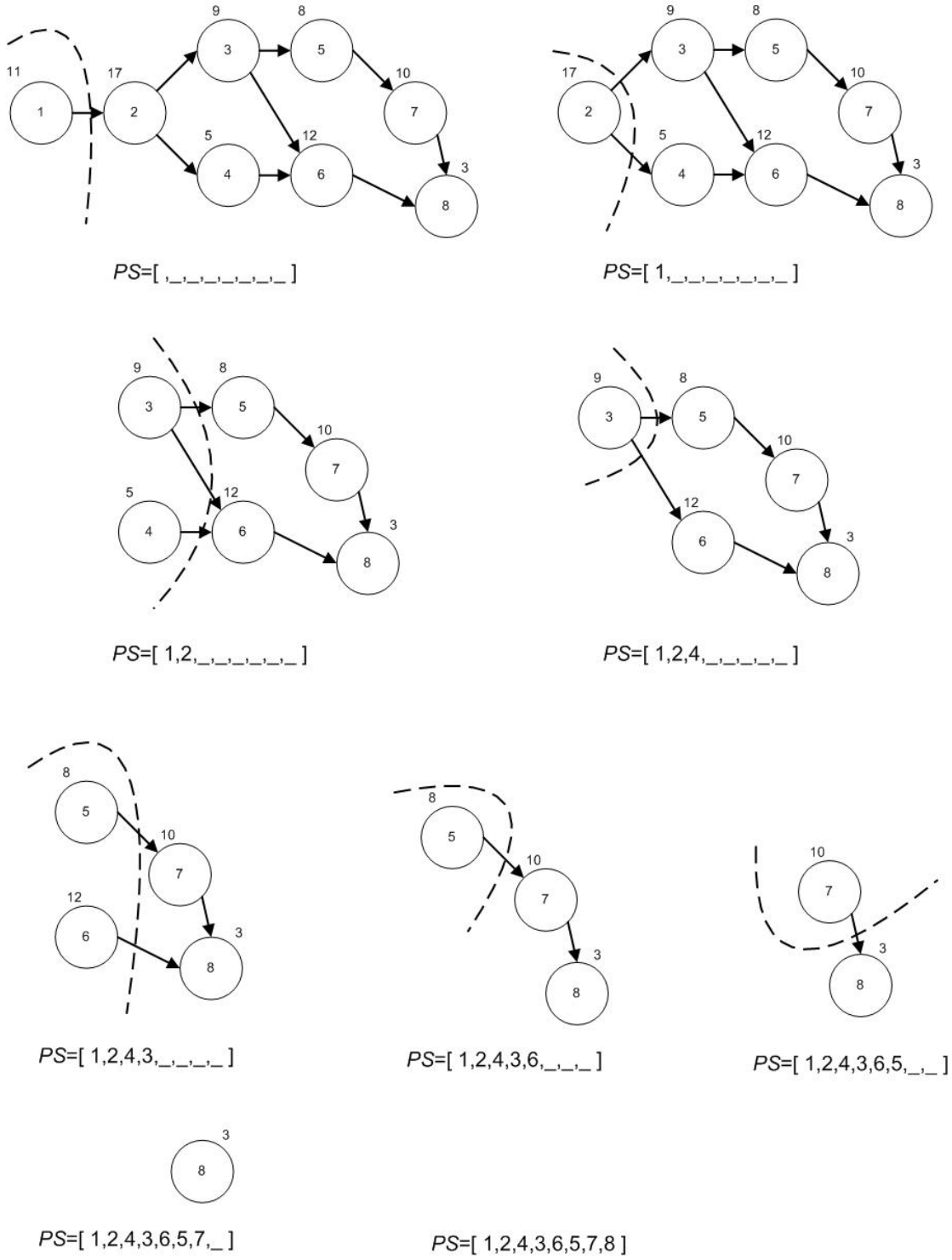

PS=[,_,_,_,_,_,_,_]
PS=[1,_,_,_,_,_,_,_]
PS=[1,2,_,_,_,_,_,_]
PS=[1,2,4,_,_,_,_,_]
PS=[1,2,4,3,_,_,_,_]
PS=[1,2,4,3,6,_,_,_]
PS=[1,2,4,3,6,5,_,_]
PS=[1,2,4,3,6,5,7,_]
PS=[1,2,4,3,6,5,7,8]

Figure 3

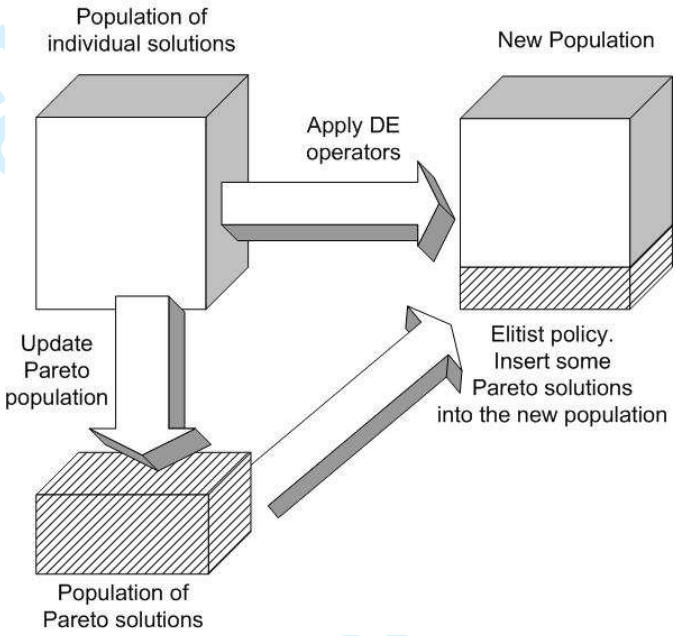


Figure 4

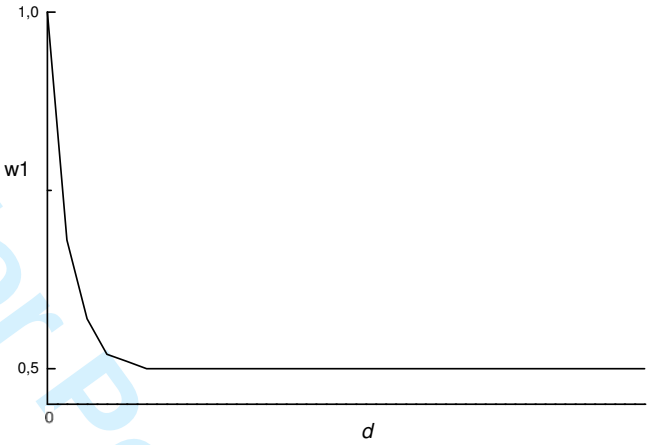
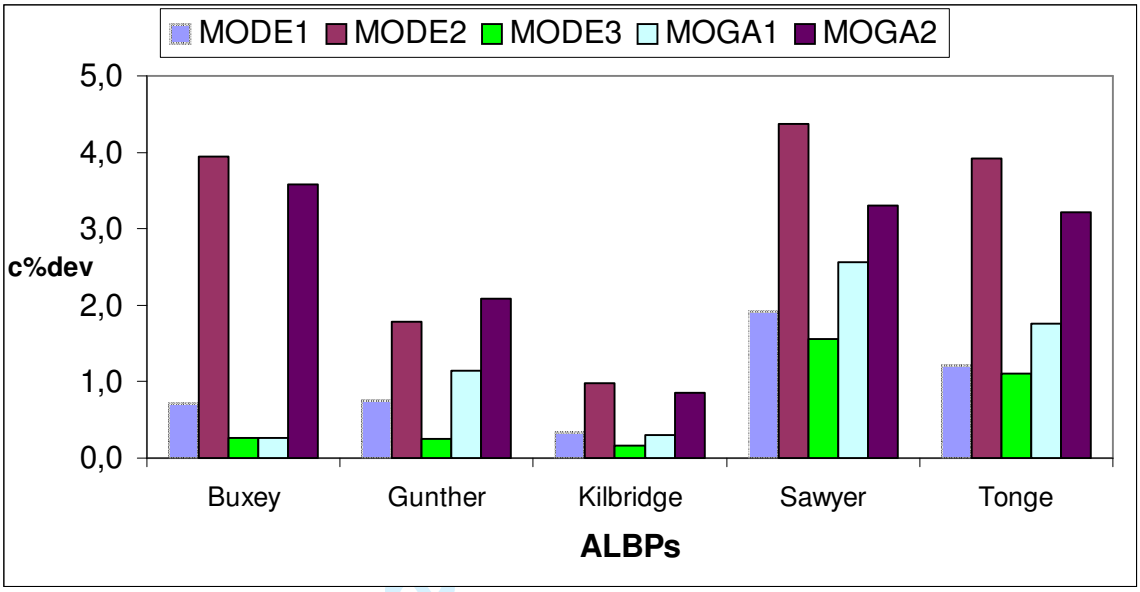
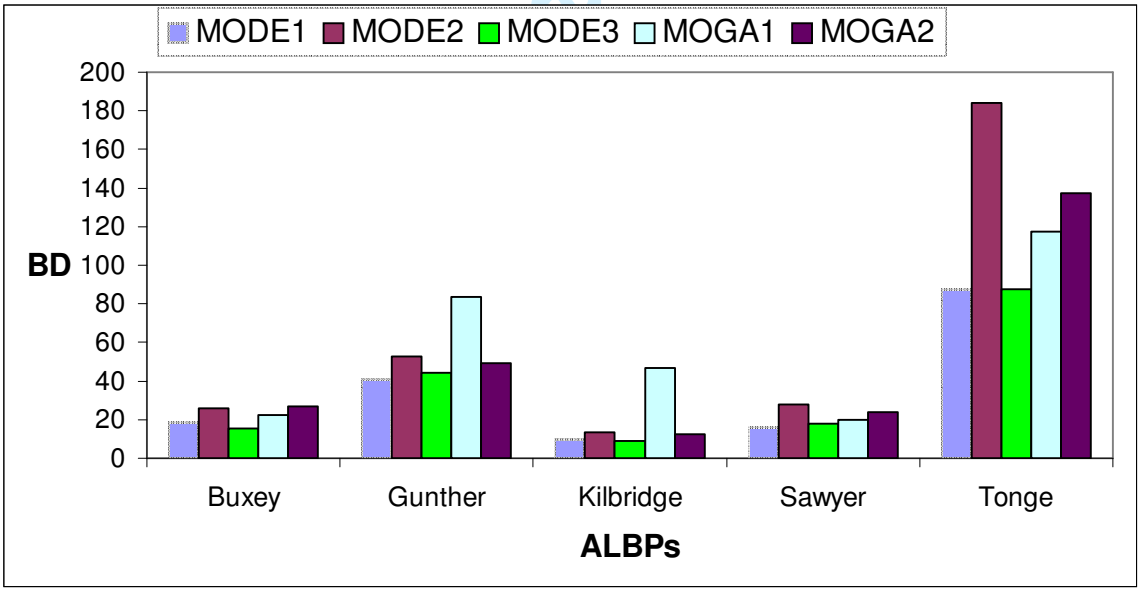


Figure 5

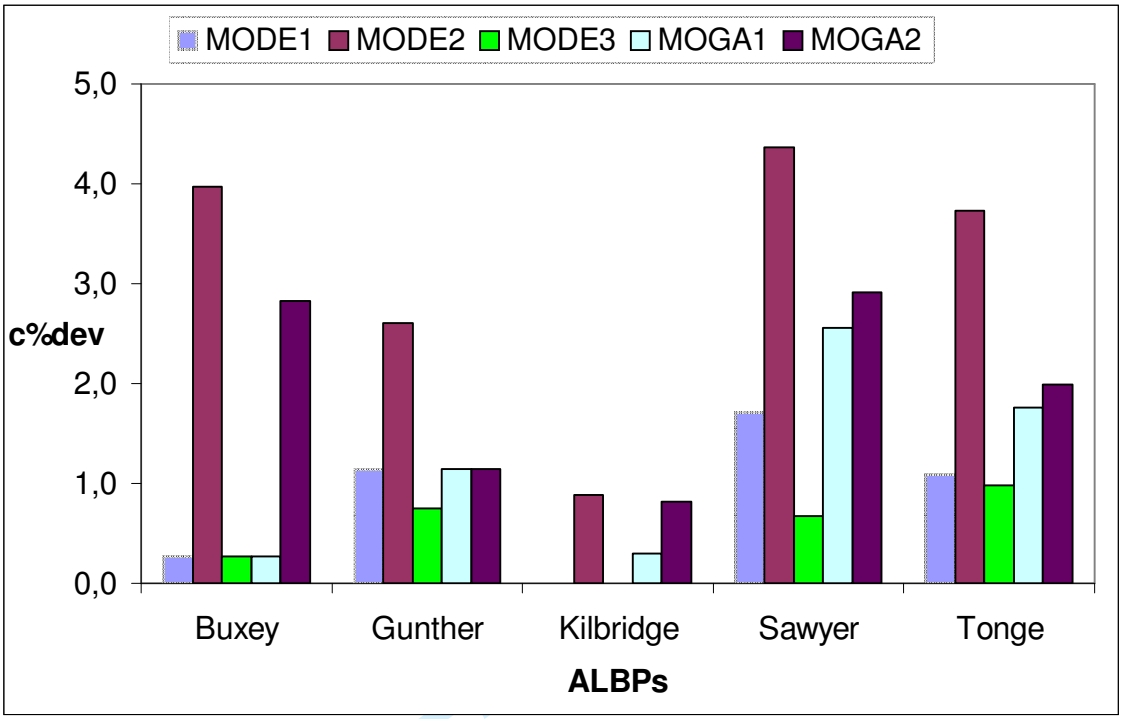

MODE1
MODE2
MODE3
MOGA1
MOGA2
5,0
4,0
3,0
2,0
1,0
0,0
c%dev
Buxey
Gunther
Kilbridge
Sawyer
Tonge
ALBPs

(a)

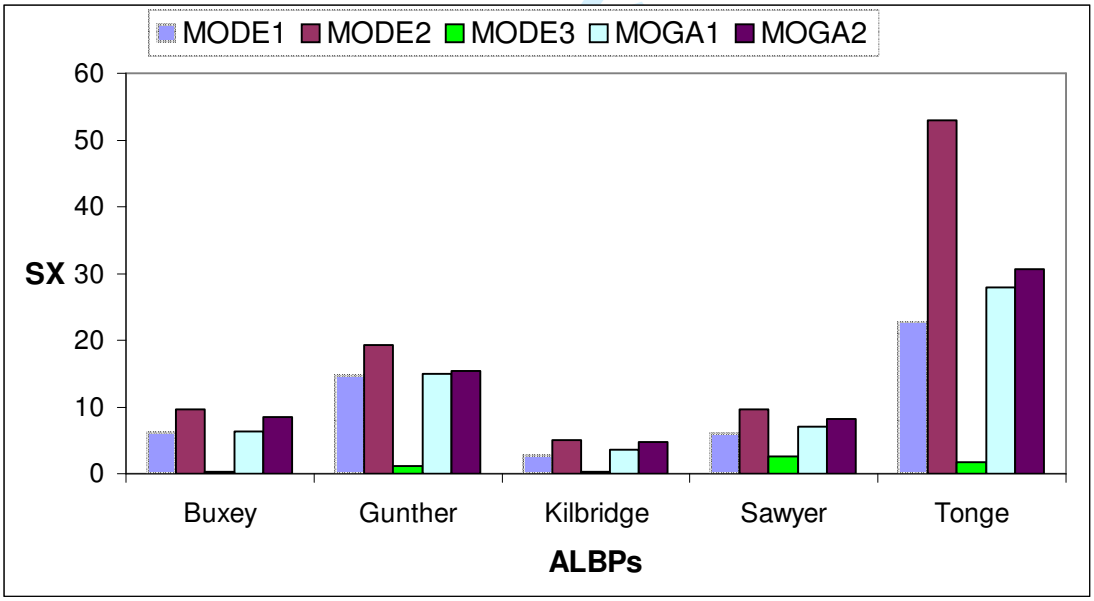

MODE1
MODE2
MODE3
MOGA1
MOGA2
200
180
160
140
120
100
80
60
40
20
0
BD
Buxey
Gunther
Kilbridge
Sawyer
Tonge
ALBPs

(b)

Figure 6

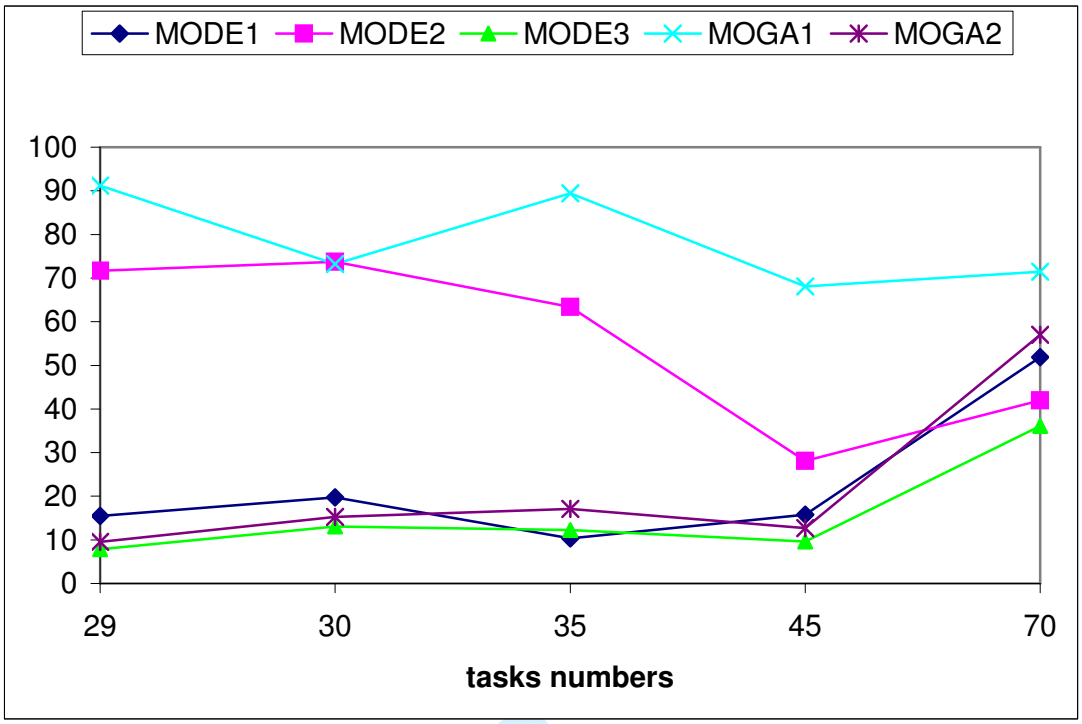

MODE1
MODE2
MODE3
MOGA1
MOGA2
5,0
4,0
3,0
2,0
1,0
0,0
c%dev
Buxey
Gunther
Kilbridge
Sawyer
Tonge
ALBPs

(a)

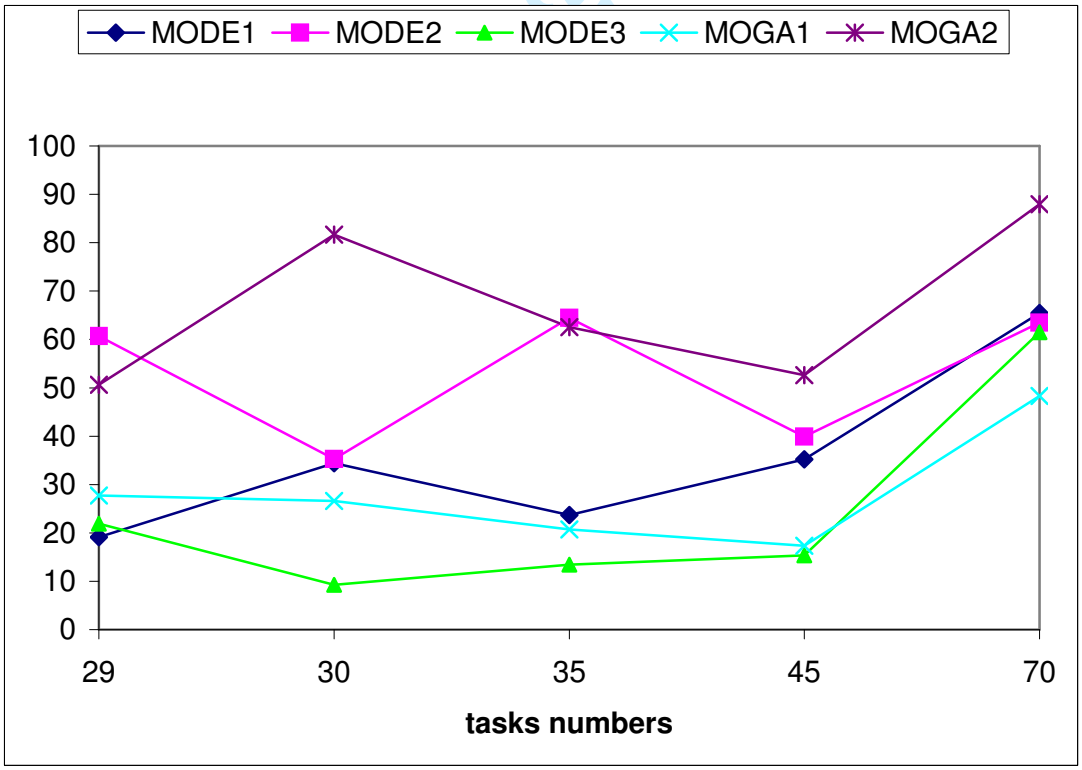

MODE1
MODE2
MODE3
MOGA1
MOGA2
60
50
40
30
20
10
0
SX
Buxey
Gunther
Kilbridge
Sawyer
Tonge
ALBPs

(b)

Figure 7


MODE1
MODE2
MODE3
MOGA1
MOGA2
100
90
80
70
60
50
40
30
20
10
0
29
30
35
45
70
tasks numbers

(a)


MODE1
MODE2
MODE3
MOGA1
MOGA2
100
90
80
70
60
50
40
30
20
10
0
29
30
35
45
70
tasks numbers

(b)

Table 1

	C_R					
$F_s \downarrow$	0.1	0.3	0.5	0.7	0.9	$C_R=\Theta\times C_R$
0.5	3.44 (55.59)	2.67 (29.38)	1.88 (47.27)	1.16 (21.15)	0.29 (25.28)	2.83 (33.36)
0.7	3.78 (48.24)	2.67 (61.60)	2.35 (27.39)	0.27 (36.46)	1.16 (7.53)	2.12 (37.34)
0.9	4.68 (53.02)	2.83 (30.31)	2.33 (49.09)	1.78 (26.99)	2.33 (33.99)	1.16 (37.90)
1.2	2.43 (56.53)	2.33 (44.79)	1.44 (55.82)	1.16 (34.56)	0.29 (30.95)	1.16 (38.73)
$F_s=\Theta\times F_s$	4.70 (49.15)	2.33 (40.69)	2.33 (38.19)	**0.27** (**34.09**)	1.16 (22.53)	2.13 (40.69)

Table 2:

ALBP	*n*	Method	c%dev	BD	Compound cost function			%effort	cpu-time (sec)
					best	worst	mean		
Buxey	29	MODE1	0.712	18.250	26.027	31.258	28.412	15.483	2.831
		MODE2	3.936	25.625	22.256	37.462	28.412	71.690	9.355
		MODE3	0.266	15.252	33.483	33.483	33.483	7.828	1.954
		MOGA1	2.226	22.500	27.571	57.824	33.483	91.172	3.133
		MOGA2	3.579	26.750	33.483	36.037	34.714	9.517	0.401
Sawyer	30	MODE1	1.922	15.750	24.641	31.258	29.303	19.767	2.804
		MODE2	4.372	27.625	21.727	36.037	29.303	73.767	9.526
		MODE3	1.562	18.125	32.333	33.483	33.483	13.067	2.996
		MOGA1	4.272	20.125	26.778	61.500	36.037	73.200	5.981
		MOGA2	3.313	24.000	33.483	34.714	33.483	15.267	1.440
Gunther	35	MODE1	0.750	40.600	44.455	46.619	46.619	10.314	6.230
		MODE2	1.778	52.700	28.412	65.667	42.478	63.371	8.253
		MODE3	0.250	44.400	51.632	70.429	51.632	12.229	5.103
		MOGA1	7.659	83.500	44.455	82.333	54.556	89.400	4.522
		MOGA2	2.082	49.400	51.632	65.667	51.632	17.114	1.453
Kilbridge	45	MODE1	0.340	9.778	46.619	51.632	49.000	15.756	8.852
		MODE2	0.981	13.333	5.329	70.429	33.483	28.067	11.031
		MODE3	0.161	8.778	82.333	82.333	82.333	9.644	9.956
		MOGA1	6.851	47.000	49.000	99.000	57.824	68.067	6.848
		MOGA2	0.860	12.667	82.333	82.333	82.333	12.689	1.241
Tonge	70	MODE1	1.213	87.609	199.000	249.000	199.000	51.900	119.889
		MODE2	3.919	183.930	82.333	332.333	199.000	42.000	142.365
		MODE3	1.109	87.700	249.000	249.000	249.000	36.100	98.719
		MOGA1	2.050	117.261	199.000	499.000	249.000	71.429	59.781
		MOGA2	3.217	137.435	54.556	110.111	89.909	52.600	31.642

Table 3

ALBP	*n*	Method	c%dev	SX	Compound cost function			%effort	cpu-time (sec)
					best	worst	mean		
Buxey	29	MODE1	0.266	6.211	19.833	23.390	20.739	19.10	3.773
		MODE2	3.969	9.580	9.638	31.258	18.231	60.69	8.473
		MODE3	0.266	7.252	32.333	33.483	33.483	21.90	5.094
		MOGA1	0.266	6.276	19.833	46.619	25.316	27.72	3.488
		MOGA2	2.831	8.513	6.937	34.714	18.231	50.62	4.421
Sawyer	30	MODE1	1.712	5.997	19.833	24.000	21.222	34.40	8.414
		MODE2	4.372	9.704	9.870	29.303	18.608	35.27	9.118
		MODE3	0.669	6.911	32.333	33.483	33.483	9.27	2.678
		MOGA1	2.562	7.002	20.277	40.667	24.641	26.61	3.373
		MOGA2	2.910	8.161	6.194	33.483	17.868	81.67	2.724
Gunther	35	MODE1	1.139	14.717	34.714	40.667	36.037	23.69	6.606
		MODE2	2.610	19.301	19.408	49.000	31.258	64.49	10.755
		MODE3	0.750	18.454	51.632	70.429	57.824	13.46	2.767
		MOGA1	1.139	14.902	33.483	75.923	37.462	20.74	9.778
		MOGA2	1.139	15.424	9.204	57.824	30.250	62.57	5.031
Kilbridge	45	MODE1	0.000	2.796	42.478	46.619	44.455	35.18	17.392
		MODE2	0.881	5.027	4.556	65.667	26.778	39.91	37.960
		MODE3	0.000	3.786	82.333	82.333	82.333	15.33	10.504
		MOGA1	0.302	3.572	42.478	89.909	49.000	17.36	13.257
		MOGA2	0.820	4.756	2.289	75.923	29.303	52.64	8.587
Tonge	70	MODE1	1.093	22.800	141.857	199.000	165.667	65.46	235.756
		MODE2	3.726	52.931	46.619	249.000	141.857	63.44	241.623
		MODE3	0.977	23.274	249.000	249.000	249.000	61.49	225.375
		MOGA1	1.755	27.858	165.667	332.333	199.000	48.33	202.106
		MOGA2	1.989	30.690	14.873	249.000	124.000	87.96	99.337